AUEB M.Sc. in Data Science (part-time)

Τρίμηνο: Εαρινό 2019

Μάθημα: Numerical Optimization and Large Scale Linear Algebra

Διδάσκων: Παρ. Βασσάλος

Φοιτητής: Σπύρος Πολίτης (p3351814)

ΑΣΚΗΣΗ 2

Notes

For problems 1-4, we have:

- K is a nxn matrix
- f is a vector
- g is a vector
- α is a scalar

Problem 1

Show that equation

$$min_{\,\mathbf{f}} \left\{ \|\mathbf{g} - \mathbf{K}\mathbf{f}\|_2^2 + \alpha^2 \|\mathbf{f}\|_2^2 \right\} \tag{1.1}$$

is equivalent to the linear least-squares problem

$$min_{\,\mathbf{f}} \left\| \begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{K} \\ \alpha\mathbf{I} \end{bmatrix} \mathbf{f} \right\|_2^2 \tag{1.2}$$

Answer

Viewing (1.2) represented as

$$\begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{x_1} \\ 0 \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{K}\mathbf{f} \\ \alpha\mathbf{f} \end{bmatrix} = \begin{bmatrix} \mathbf{y_1} \\ \alpha\mathbf{y_2} \end{bmatrix}$$

with $x_1, y_1 \in R^l$, $y_2 \in R^m$, such that $l + m = n$, and α a scalar, we get the following:

$$\begin{gathered} \left\| \begin{bmatrix} \mathbf{x_1} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{y_1} \\ \alpha\mathbf{y_2} \end{bmatrix} \right\|_2^2 = \\ \left\| \begin{bmatrix} \mathbf{x_1} - \mathbf{y_1} \\ -\alpha\mathbf{y_2} \end{bmatrix} \right\|_2^2 = \\ \left(\sqrt{(\mathbf{x_1} - \mathbf{y_1})^2 + (-\alpha\mathbf{y_2})^2} \right)^2 = \\ (\mathbf{x_1} - \mathbf{y_1})^2 + (-\alpha\mathbf{y_2})^2 = \\ (\mathbf{x_1} - \mathbf{y_1})^2 + \alpha^2 {\mathbf{y_2}}^2 = \end{gathered}$$

(substituting $\mathbf{x_1}$, $\mathbf{y_1}$, $\mathbf{y_2}$ and because they are vectors)

$$\|\mathbf{g} - \mathbf{K}\mathbf{f}\|_2^2 + \alpha^2 \|\mathbf{f}\|_2^2 \tag{1.3}$$

Clearly, (1.3) is equal to (1.1).

Problem 2

Show that if K has a singular value decomposition $\mathbf{K} = \mathbf{U\Sigma V^T}$, then equation

$$min_{\,\mathbf{f}} \left\| \begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{K} \\ \alpha \mathbf{I} \end{bmatrix} \mathbf{f} \right\|_2^2 \tag{2.1}$$

can be transformed to the equivalent least squares problem

$$min_{\,\mathbf{g}} \left\| \begin{bmatrix} \hat{\mathbf{g}} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{\Sigma} \\ \alpha \mathbf{I} \end{bmatrix} \hat{\mathbf{f}} \right\|_2^2 \tag{2.2}$$

where $\hat{\mathbf{f}} = \mathbf{V^T f}$ and $\hat{\mathbf{g}} = \mathbf{U^T g}$.

Answer

From problem 1, eq. (1.3), we can deduce that (2.2) can be written as:

$$\begin{gathered} \left\| \hat{\mathbf{g}} - \mathbf{\Sigma} \hat{\mathbf{f}} \right\|_2^2 + \alpha^2 \left\| \hat{\mathbf{f}} \right\|_2^2 = \\ (\hat{\mathbf{g}} - \mathbf{\Sigma} \hat{\mathbf{f}})^T (\hat{\mathbf{g}} - \mathbf{\Sigma} \hat{\mathbf{f}}) + \alpha^2 \hat{\mathbf{f}}^T \hat{\mathbf{f}} = \\ \hat{\mathbf{g}}^T \hat{\mathbf{g}} - \hat{\mathbf{g}}^T \mathbf{\Sigma} \hat{\mathbf{f}} - (\mathbf{\Sigma} \hat{\mathbf{f}})^T \hat{\mathbf{g}} + (\mathbf{\Sigma} \hat{\mathbf{f}})^T \mathbf{\Sigma} \hat{\mathbf{f}} + \alpha^2 \hat{\mathbf{f}}^T \hat{\mathbf{f}} = \\ \hat{\mathbf{g}}^T \hat{\mathbf{g}} - 2 \hat{\mathbf{g}}^T \mathbf{\Sigma} \hat{\mathbf{f}} + \hat{\mathbf{f}}^T \mathbf{\Sigma}^T \mathbf{\Sigma} \hat{\mathbf{f}} + \alpha^2 \hat{\mathbf{f}}^T \hat{\mathbf{f}} \end{gathered} \tag{2.3}$$

Breaking up (2.3) we get the following equations and setting $\hat{\mathbf{f}} = \mathbf{V^T f}$ and $\hat{\mathbf{g}} = \mathbf{U^T g}$ we get the following:

$$\hat{\mathbf{g}}^T \hat{\mathbf{g}} = (\mathbf{U}^T \mathbf{g})^T \mathbf{U}^T \mathbf{g} = \mathbf{g}^T \mathbf{U} \mathbf{U}^T \mathbf{g} = \mathbf{g}^T \mathbf{g} \tag{2.4}$$

Note: on (2.4) $\mathbf{U}\mathbf{U}^T = \mathbf{I}$, because $\mathbf{U}$ is orthogonal.

$$\begin{gathered} -2 \hat{\mathbf{g}}^T \mathbf{\Sigma} \hat{\mathbf{f}} = -2 (\mathbf{U}^T \mathbf{g})^T \mathbf{\Sigma} \mathbf{V}^T \mathbf{f} = -2 \mathbf{g}^T \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \mathbf{f} = -2 \mathbf{g}^T \mathbf{K} \mathbf{f} \\ \hat{\mathbf{f}}^T \mathbf{\Sigma}^T \mathbf{\Sigma} \hat{\mathbf{f}} = (\mathbf{V}^T \mathbf{f})^T \mathbf{\Sigma}^T \mathbf{\Sigma} \mathbf{V}^T \mathbf{f} = \mathbf{f}^T \mathbf{V} \mathbf{\Sigma}^T \mathbf{\Sigma} \mathbf{V}^T \mathbf{f} = \mathbf{f}^T \mathbf{V} \mathbf{\Sigma}^T \mathbf{U}^T \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \mathbf{f} = \mathbf{f}^T \mathbf{K}^T \mathbf{K} \mathbf{f} \end{gathered} \tag{2.5}$$

Note: on (2.6) we have set $\mathbf{U}^T\mathbf{U}$ on step 4, as a substitution for $\mathbf{I}$, since $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. We did this because we know that $\mathbf{K} = \mathbf{U\Sigma V}^T$ and we need to arrive to such a factorization that will allow us to substitute $\mathbf{U\Sigma V}^T$ with $\mathbf{K}$.

$$\alpha^2 \hat{\mathbf{f}}^T \hat{\mathbf{f}} = \alpha^2 (\mathbf{V}^T \mathbf{f})^T \mathbf{V}^T \mathbf{f} = \alpha^2 \mathbf{f}^T \mathbf{V} \mathbf{V}^T \mathbf{f} = \alpha^2 \mathbf{f}^T \mathbf{f} \tag{2.7}$$

Note: on (2.7) $\mathbf{V}\mathbf{V}^T = \mathbf{I}$, because $\mathbf{V}$ is orthogonal.

Substituting (2.4), (2.5), (2.6) and (2.7) back into (2.3), we get:

$$\mathbf{g}^T \mathbf{g} - 2 \mathbf{g}^T \mathbf{K} \mathbf{f} + \mathbf{f}^T \mathbf{K}^T \mathbf{K} \mathbf{f} + \alpha^2 \mathbf{f}^T \mathbf{f} = (\mathbf{g} - \mathbf{K} \mathbf{f})^T (\mathbf{g} - \mathbf{K} \mathbf{f}) + \alpha^2 \mathbf{f}^T \mathbf{f} = \|\mathbf{g} - \mathbf{K} \mathbf{f}\|_2^2 + \alpha^2 \|\mathbf{f}\|$$

Since we know from problem 1 that (2.8) is equivalent to (1.2), we have shown that (2.2) is equivalent to (2.1).

Problem 3

Determine a formula for the solution of equation

$$min_{\,\mathbf{g}} \left\| \begin{bmatrix} \hat{\mathbf{g}} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{\Sigma} \\ \alpha \mathbf{I} \end{bmatrix} \hat{\mathbf{f}} \right\|_2^2 \tag{2.2}$$

where $\hat{\mathbf{f}} = \mathbf{V^T f}$ and $\hat{\mathbf{g}} = \mathbf{U^T g}$.

Hint: you should set the drivative of the minimization function to zero and solve for $\hat{\mathbf{f}}$.

Answer

From (2.3) we have:

$$\begin{gathered} min_{\,\mathbf{g}} \left\| \hat{\mathbf{g}} - \mathbf{\Sigma} \hat{\mathbf{f}} \right\|_2^2 + \alpha^2 \left\| \hat{\mathbf{f}} \right\|_2^2 = \\ min_{\,\mathbf{g}} \left(\left(\sqrt{\sum_{i=1}^{n} (\hat{g}_i - \sigma_i \hat{f}_{\,i})^2} \right)^2 + \alpha^2 \left(\sqrt{\sum_{i=1}^{n} \hat{f}_{\,i}^{\,2}} \right)^2 \right) = \\ min_{\,\mathbf{g}} \left(\sum_{i=1}^{n} (\hat{g}_i - \sigma_i \hat{f}_{\,i})^2 + \alpha^2 \sum_{i=1}^{n} \hat{f}_{\,i}^{\,2} \right) \end{gathered} \tag{3.1}$$

Taking the partial derivative of (3.1) w.r.t. $\hat{f}$ we have:

$$\frac{\partial}{\partial \hat{f}} = 2(\hat{g}_i - \sigma_i \hat{f}_{\,i})(-\sigma_i) + 2\alpha^2 \hat{f}_{\,i} = -2\sigma_i \hat{g}_i + 2\sigma_i^2 \hat{f}_{\,i} + 2\alpha^2 \hat{f}_{\,i} \tag{3.2}$$

Setting (3.2) to zero and solving for $\hat{f}_{\,i}$ we get:

$$
\begin{aligned}
-2\sigma_i \hat{g}_i + 2\sigma_i^2 \hat{f}_i + 2\alpha^2 \hat{f}_i = 0 \implies \\
-\sigma_i \hat{g}_i + \sigma_i^2 \hat{f}_i + \alpha^2 \hat{f}_i = 0 \implies \\
\hat{f}_i(\sigma_i^2 + \alpha^2) = \sigma_i \hat{g}_i \implies \\
\hat{f}_i = \frac{\sigma_i \hat{g}_i}{\sigma_i^2 + \alpha^2}
\end{aligned} \tag{3.3}
$$

Problem 4

Show that the solution to the problem

$$min_{\mathbf{f}} \left\{ \|\mathbf{g} - \mathbf{K}\mathbf{f}\|_2^2 \right\} \tag{4.1}$$

is

$$\mathbf{f}_{ls} = \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^T\mathbf{g} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T\mathbf{g}}{\sigma_i}\mathbf{v}_i \tag{4.2}$$

where u_i is the ith column column of $\mathbf{U}$ and $\mathbf{v}_i$ is the ith column of $\mathbf{V}$.

Answer

Let's visualize the matrices, vectors and their operations, as stated in the problem:

$$
\mathbf{U} = \begin{bmatrix} | & | & \cdots & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & \cdots & | \end{bmatrix}, \mathbf{U}^T = \begin{bmatrix} — & u_1^T & — \\ — & u_2^T & — \\ \vdots & \vdots & \vdots \\ — & u_n^T & — \end{bmatrix}
$$

$$
\mathbf{V} = \begin{bmatrix} | & | & \cdots & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & \cdots & | \end{bmatrix}, \mathbf{V}^T = \begin{bmatrix} — & v_1^T & — \\ — & v_2^T & — \\ \vdots & \vdots & \vdots \\ — & v_n^T & — \end{bmatrix}
$$

From problem (2), we know that:

$$\hat{\mathbf{g}} = \mathbf{U^T g} \implies \hat{g}_i = \mathbf{u}_i^T \mathbf{g} \tag{4.3}$$

$$\hat{\mathbf{f}} = \mathbf{V^T f} \implies \hat{f}_i = \mathbf{v}_i^T \mathbf{g} \tag{4.4}$$

From problem (3), equation (3.3), we expressed the least squares problem as:

$$\hat{f}_i = \frac{\sigma_i \hat{g}_i}{\sigma_i^2 + \alpha^2} \tag{4.5}$$

Since we are been asked to evaluate

$$min_{\mathbf{f}} \left\{ \|\mathbf{g} - \mathbf{K f}\|_2^2 \right\}$$

we can infer that no regularization is considered, (i.e. for $\alpha = 0$). Hence the above becomes:

$$\hat{f}_i = \frac{\sigma_i \hat{g}_i}{\sigma_i^2} \implies \hat{f}_i = \frac{\hat{g}_i}{\sigma_i} \implies \hat{\mathbf{f}} = \sum_{i=1}^{n} \frac{\hat{g}_i}{\sigma_i} \tag{4.6}$$

Therefore, from (4.3), (4.4), (4.6) and because $\mathbf{V}$ is orthogonal, we have:

$$\begin{gathered}
\mathbf{V}^T \mathbf{f} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{g}}{\sigma_i} \implies \\
\mathbf{f} = (\mathbf{V}^T)^{-1} \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{g}}{\sigma_i} \implies \\
\mathbf{f} = \mathbf{V} \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{g}}{\sigma_i} \implies \\
\mathbf{f} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T \mathbf{g}}{\sigma_i} \mathbf{v}_i
\end{gathered}$$

Problem 5

Write a program that takes matrices $\mathbf{A}$, $\mathbf{B}$ and image $\mathbf{G}$ and computes approximations to image F using Tikhonov regularization and Truncated SVD. For easch of these two algorithms, experiment to find the value of the regularization parameter (a for Tikhonov and p for TSVD) that gives the clearest image. In the file of the project you will find the necessary matrices.

Answer

Introductory problem statement and theory

The generic form of the problem we are called upon to solve is:

$$\mathbf{g} = \mathbf{K}\mathbf{f} + \eta$$

where:

$\mathbf{g}$: the n^2 vector of observed image pixels.

$\mathbf{K}$: a nxn ill-conditioned degradation matrix representing the blurring operator applied to the original, unknown image.

$\mathbf{f}$: approximation of the original image as a n^2 vector.

η: random noise

$\mathbf{A}$, $\mathbf{B}$ are given as inputs to the problem an they represent the operands of the Kronecker product $\mathbf{K} = \mathbf{A} \otimes \mathbf{B}$ that produces matrix $\mathbf{K}$. This product is too large to be used directly in SVD. In fact, it is of the order $n^2 * n^2 = n^4$. It quickly becomes evident that we need a more efficient method, by way of computing partial Kronecker products.

Utilising the following theorem:

If

$$\mathbf{A} = \mathbf{U}_A \mathbf{\Sigma}_A \mathbf{V}_A^T$$
$$\mathbf{B} = \mathbf{U}_B \mathbf{\Sigma}_B \mathbf{V}_B^T$$

then

$$\mathbf{K} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$$

with

$$\mathbf{U} = \mathbf{U}_A \otimes \mathbf{U}_B$$
$$\mathbf{\Sigma} = \mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B$$
$$\mathbf{V} = \mathbf{V}_A \otimes \mathbf{V}_B$$

we can tell that, in order to compute the SVD of $\mathbf{K}$, we need to reconstruct $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ from the respective Kronecker products of the components of the SVDs of matrices $\mathbf{A}$, $\mathbf{B}$. In other words, the components of the SVDs of the given input matrices $\mathbf{A}$, $\mathbf{B}$ are the factors of the Kronecker products of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$.

Applying singular value decomposition to the matrix A, we have

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

$\mathbf{\Sigma}$ is a diagonal matrix of singular values of the matrix $\mathbf{A}$, whose non-zero diagonal elements $\lambda_1, \lambda_2, \cdots, \lambda_i$ are, without loss of generality, assumed to be positive and arranged in decreasing order i.e.
$\lambda_1 \geq \lambda_2 \geq \lambda_i > 0$.

In [1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

In [2]:
```
# Import required packages.
import sys
import multiprocessing as mp
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
# Import required classes.
sys.path.append('code')

from DataHelper import DataHelper
from VisualizationHelper import VisualizationHelper
```

In [4]:
```
# Initialize required classes.
data_helper = DataHelper()
visualization_helper = VisualizationHelper()
```

In [5]:
```
# Define matrix size.
n = 256
```

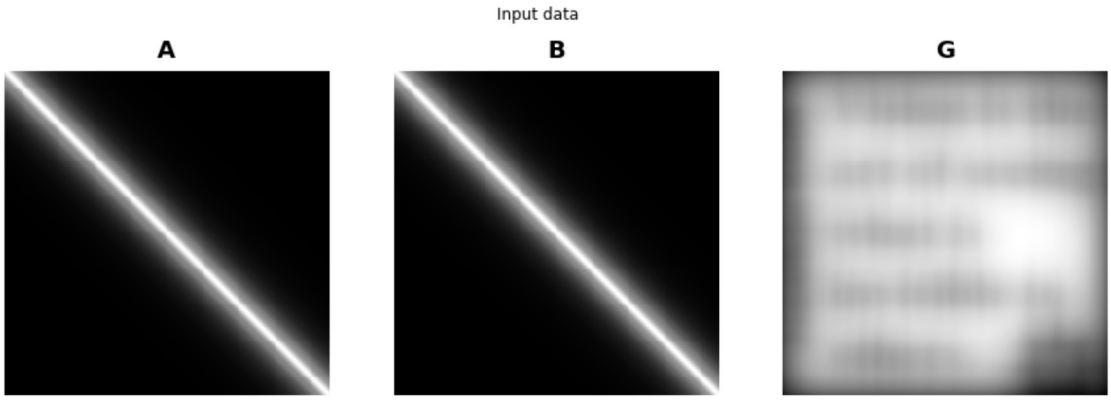
In [6]:
```
%matplotlib inline
```

We ingest the data provided for the assignment and tranform it to required matrices A, B, G:

In [34]:
```
A, B, G = data_helper.ingest_data(file_path = 'data', n = 256)
```

Let's take a look at our input data:

In [76]:
```
plot_main_title_font = {
    'family': 'sans serif',
    'color':  'black',
    'weight': 'bold',
    'size': 18
}

# Setup the plot
fig, ax = plt.subplots(nrows = 1, ncols = 3, figsize = (15, 5))

fig.suptitle('Input data', fontdict = plot_main_title_font)

ax[0].axis('off')
ax[0].set_title(label = 'A', loc = 'center', fontdict = plot_main_title
_font, pad = 10)
ax[0].imshow(A, cmap = plt.cm.gray)

ax[1].axis('off')
ax[1].set_title(label = 'B', loc = 'center', fontdict = plot_main_title
_font, pad = 10)
ax[1].imshow(B, cmap = plt.cm.gray)

ax[2].axis('off')
ax[2].set_title(label = 'G', loc = 'center', fontdict = plot_main_title
_font, pad = 10)
ax[2].imshow(G, cmap = plt.cm.gray)

pass;
```



Truncated SVD

In order to compute the truncated SVD reconstruction of $\mathbf{F}$, we utilized the solution to problem (4.1), specifically its equivalentl expression (4.2).

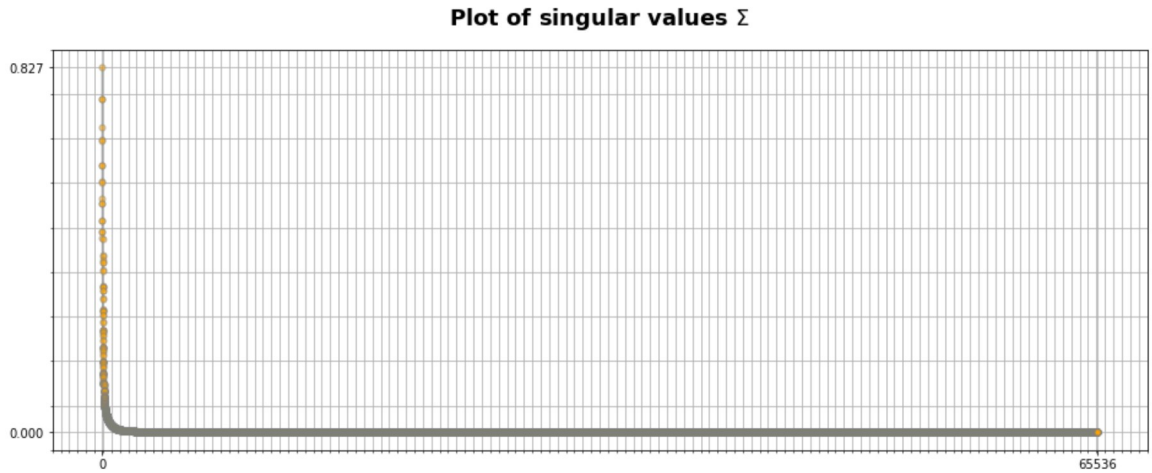
To achieve the computation of (4.2), we implemented the following process:

- Computed the quantities $\mathbf{U_A}$, $\mathbf{\Sigma_A}$, $\mathbf{V_A^T}$ by performing SVD on $\mathbf{A}$. Similarly, we computed the quantities $\mathbf{U_B}$, $\mathbf{\Sigma_B}$, $\mathbf{V_B^T}$ by performing SVD on $\mathbf{B}$.
- Computed the Kronecker product of ($\mathbf{\Sigma_A}$, $\mathbf{\Sigma_B}$), an efficient computation since both are vectors of size n^2. The reason for this computation is to retrieve the singular values of $\mathbf{K}$.
- The Kronecker product of ($\mathbf{\Sigma_A}$, $\mathbf{\Sigma_B}$) was then sorted in descending order, thereby allowing us to retrieve the p-first, most significant singular values of $\mathbf{K}$.
- Computing $\hat{f}$ is then achieved by applying (4.2), making sure to select such columns of $\mathbf{U_A}$, $\mathbf{V_A}$ and $\mathbf{U_B}$, $\mathbf{V_B}$, such that they correspond to the ordered index of the Kronecker product of ($\mathbf{\Sigma_A}$, $\mathbf{\Sigma_B}$), as if $\mathbf{\Sigma}$ was represented in diagonal matrix form (not computationally feasible, since such a representation would yield a $n^2 \times n^2$ matrix.).

In order to pick a good candidate value for p, we need to visualize the order singular values so that we can get a visual feel of their weight:

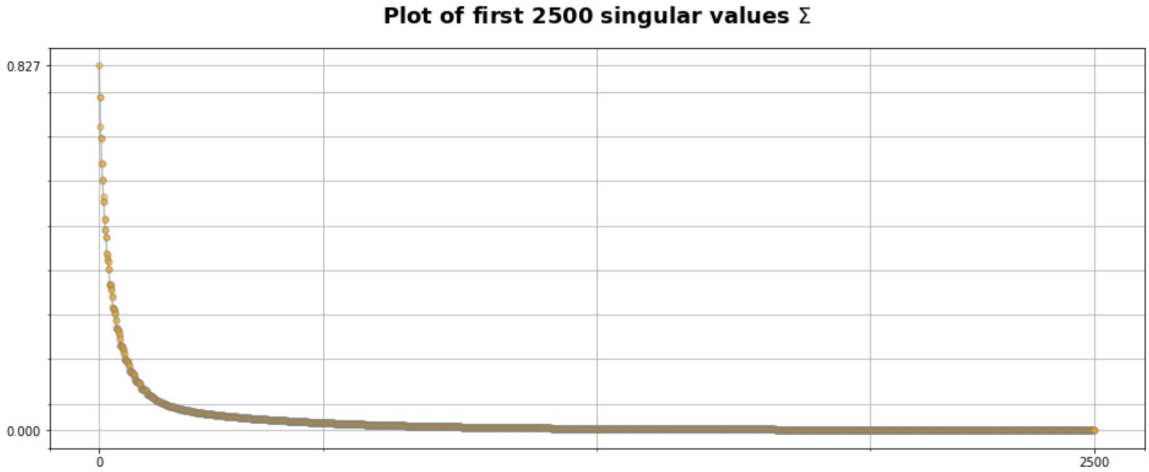
In [9]:
```
# Compute SVD of A, B.
U_A, S_A, V_A_T = np.linalg.svd(A)
U_B, S_B, V_B_T = np.linalg.svd(B)

S = np.flip(np.sort(np.kron(S_A, S_B)))
```

In [10]:
```
visualization_helper.plot_points(S, size = (16, 6), title = 'Plot of si
ngular values $\Sigma$')

pass;
```



let's zoom in a little at the "elbow" of the plot:

In [11]:
```
p = 2500

visualization_helper.plot_points(S[0:p], size = (16, 6), title = 'Plot
of first ' + str(p) + ' singular values $\Sigma$')

pass;
```



We estimate the the bulk of the most sognificant singular values of $\mathbf{\Sigma}$ are the first 2500. We shall therefore limit our search space to within 500 and 6000 singular values, so as to have a visual validation of the method.

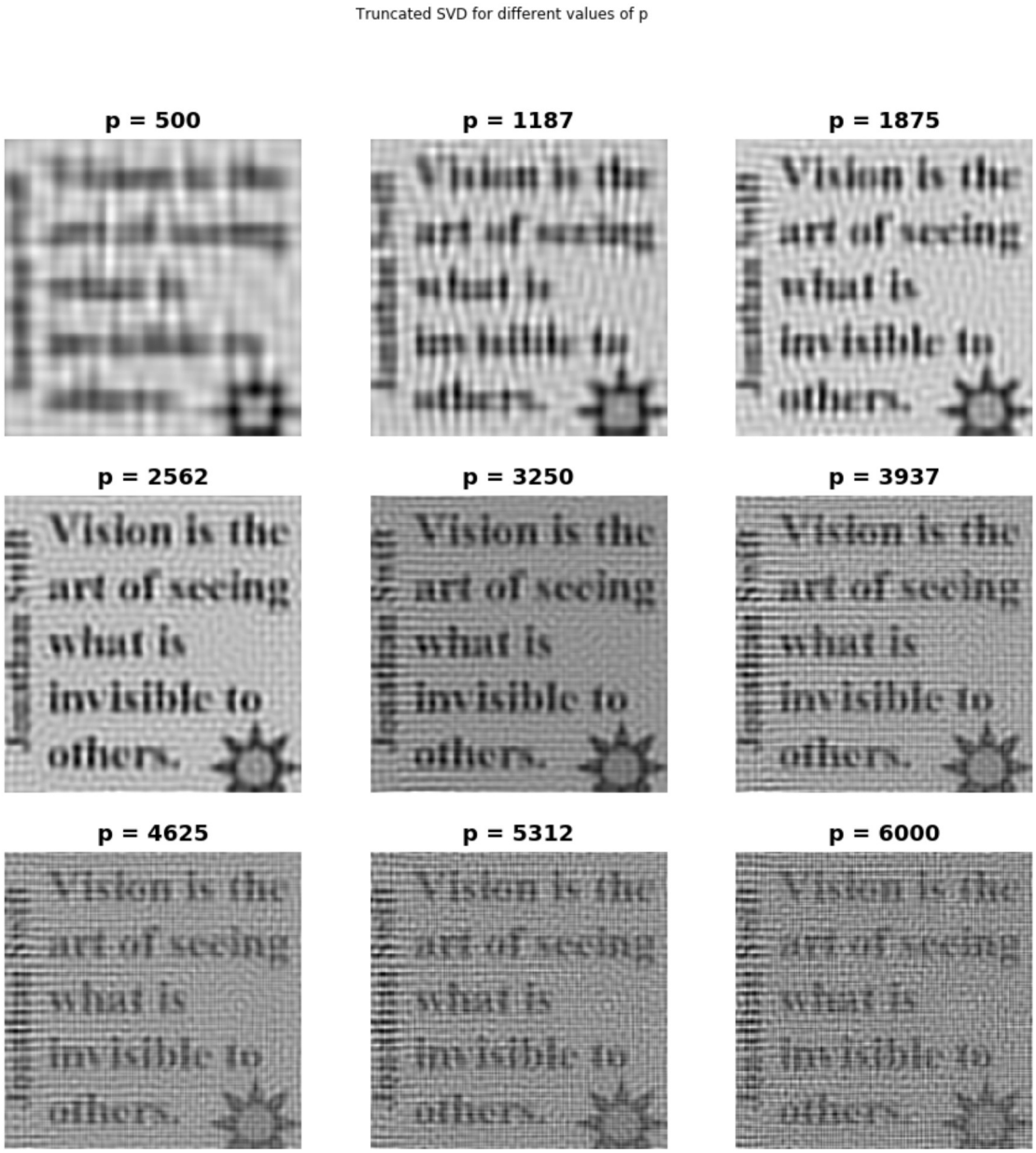
Parallel computation of Truncated SVD

This approach seeks to parallelize the inherently single-process computation restrictions of Python in a way such that multi-core CPUs can be utilized to their full potential.

A word of warning: the code has been tested on *Linux* (*Ubuntu 18.04.1*) and *Windows 10*, not on *MacOS*. If, for any reason, it fails to run on a platform other than *Linux* or *Windows 10*, the serial computation method should be employed.

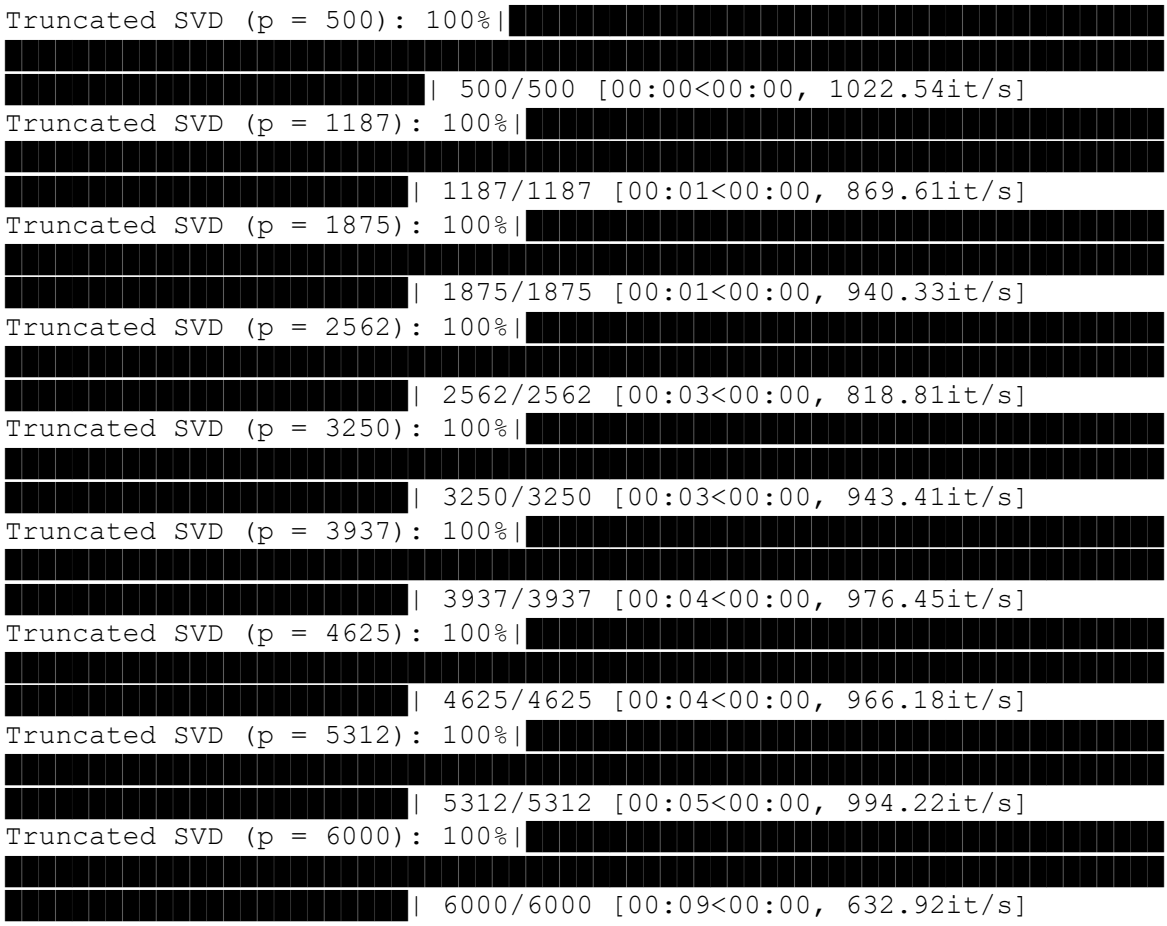
In [12]:
```
# Truncation parameters p.
p = np.linspace(500, 6000, 9).astype(np.int32)

# Spawn tasks for parallel computation.
mp_runner_truncated_svd_results = data_helper.mp_runner('TruncatedSVD',
'TruncatedSVD', p, n, A, B, G)
```

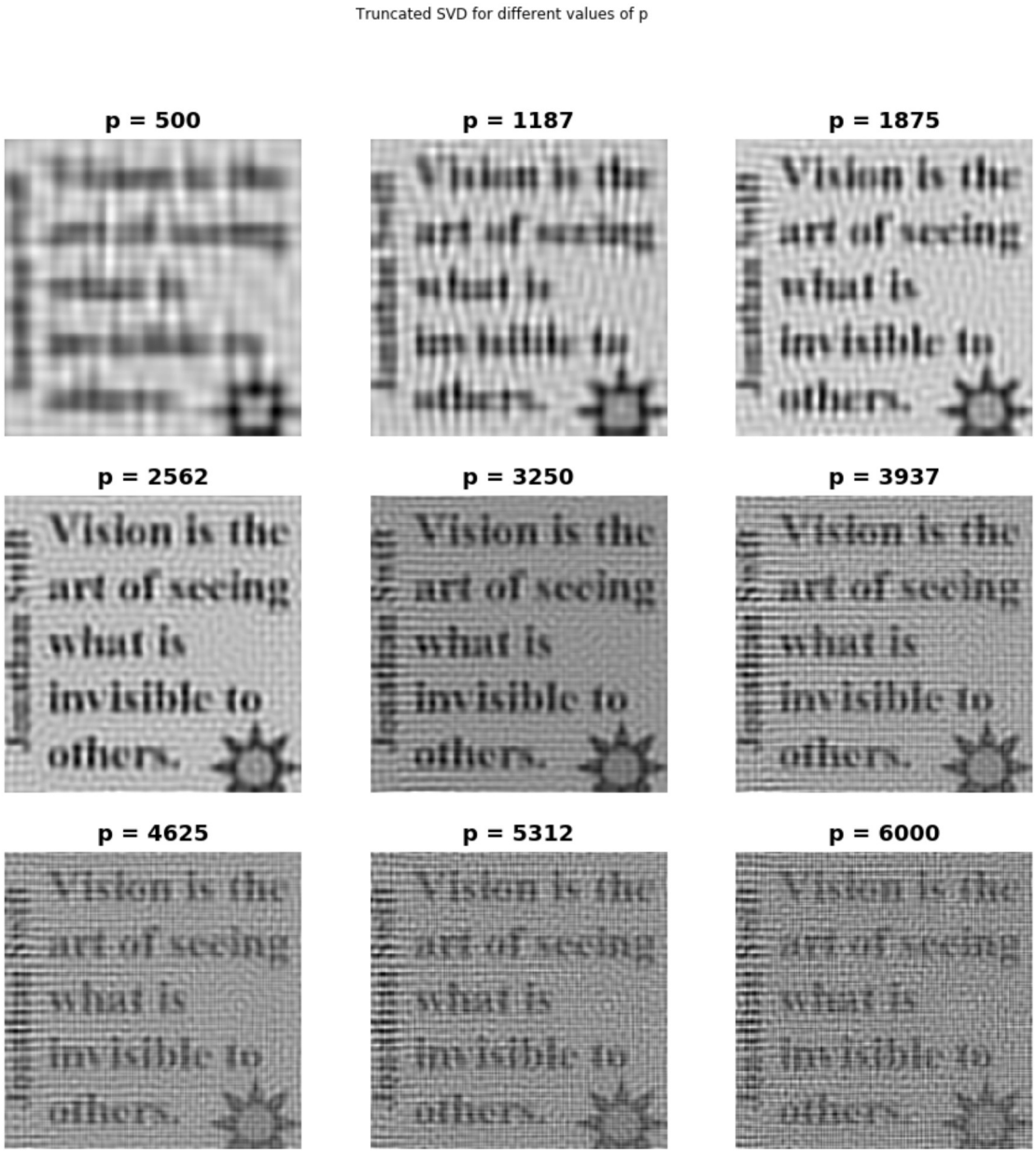
In [26]:
```
plot_main_title_font = {
    'family': 'sans serif',
    'color':  'black',
    'weight': 'bold',
    'size': 18
}

# Number of plot rows, columns
nrows = 3
ncols = 3

# Axis counters
ax_1 = 0
ax_2 = 0

# Setup the plot
fig, ax = plt.subplots(nrows = nrows, ncols = ncols, figsize = (15, 15)
)

fig.suptitle('Truncated SVD for different values of p', fontdict = plot
_main_title_font)

for i in range(0, len(mp_runner_truncated_svd_results)):
    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

    # Set graph title
    ax[ax_1, ax_2].set_title(label = 'p = ' + str(p[i]), loc = 'center'
, fontdict = plot_main_title_font, pad = 10)

    ax[ax_1, ax_2].imshow(mp_runner_truncated_svd_results[i].F, cmap =
plt.cm.gray)

# Hide unused axes.
for i in range(len(p), nrows * ncols):
    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

pass;
```



Serial computation of Truncated SVD

In [27]:
```
from TruncatedSVD import TruncatedSVD

plot_main_title_font = {
    'family': 'sans serif',
    'color':  'black',
    'weight': 'bold',
    'size': 18
}

p = np.linspace(500, 6000, 9).astype(np.int32)

# Number of plot rows, columns
nrows = 3
ncols = 3

# Axis counters
ax_1 = 0
ax_2 = 0

# Setup the plot
fig, ax = plt.subplots(nrows = nrows, ncols = ncols, figsize = (15, 15)
)

fig.suptitle('Truncated SVD for different values of p', fontdict = plot
_main_title_font)

truncated_svd = TruncatedSVD(with_progress_bar = True)

for i in range(0, len(p)):
    truncated_svd = truncated_svd.solve(n = n, A = A, B = B, G = G, p =
p[i])

    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

    # Set graph title
    ax[ax_1, ax_2].set_title(label = 'p = ' + str(p[i]), loc = 'center'
, fontdict = plot_main_title_font, pad = 10)

    ax[ax_1, ax_2].imshow(truncated_svd.F, cmap = plt.cm.gray)


# Hide unused axes.
for i in range(len(p), nrows * ncols):
    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

pass;
```


Truncated SVD (p = 500): 100%| 500/500 [00:00<00:00, 1022.54it/s]
Truncated SVD (p = 1187): 100%| 1187/1187 [00:01<00:00, 869.61it/s]
Truncated SVD (p = 1875): 100%| 1875/1875 [00:01<00:00, 940.33it/s]
Truncated SVD (p = 2562): 100%| 2562/2562 [00:03<00:00, 818.81it/s]
Truncated SVD (p = 3250): 100%| 3250/3250 [00:03<00:00, 943.41it/s]
Truncated SVD (p = 3937): 100%| 3937/3937 [00:04<00:00, 976.45it/s]
Truncated SVD (p = 4625): 100%| 4625/4625 [00:04<00:00, 966.18it/s]
Truncated SVD (p = 5312): 100%| 5312/5312 [00:05<00:00, 994.22it/s]
Truncated SVD (p = 6000): 100%| 6000/6000 [00:09<00:00, 632.92it/s]



A key difficulty in applying TSVD methods has been how to set up proper criteria to truncate the singular values of Σ. Several methods exist, including the significance test and L-curve.

(source: Truncated SVD methods for discrete linear ill-posed problems (https://academic.oup.com/gji/article-pdf/135/2/505/2338267/135-2-505.pdf (https://academic.oup.com/gji/article-pdf/135/2/505/2338267/135-2-505.pdf))

However, for the scope of this assignment, we have relied on the singular values plot and "eyeballing" the best value of p. As per these methods, it appears that the best candidate for p is 2500.

Tikhonov regularization (Regularized Least Squares optimization)

Recall that the problem we seek to solve, from (1.1) is

$$min_{\,\mathbf{f}}\ \{\|\mathbf{g}-\mathbf{K}\mathbf{f}\|_2^2+\alpha^2\|\mathbf{f}\|_2^2\}$$

We seek a good value for parameters α, such that it limits the size of the coefficients by adding a penalty equal to the L2 norm of the magnitude of coefficients (i.e. performing L2 regularization).

In effect, we seek to formulate a less noise-sensitive approach to the Least Squares problem.

In order to solve the problem, we will use its equivalent (3.3) representation

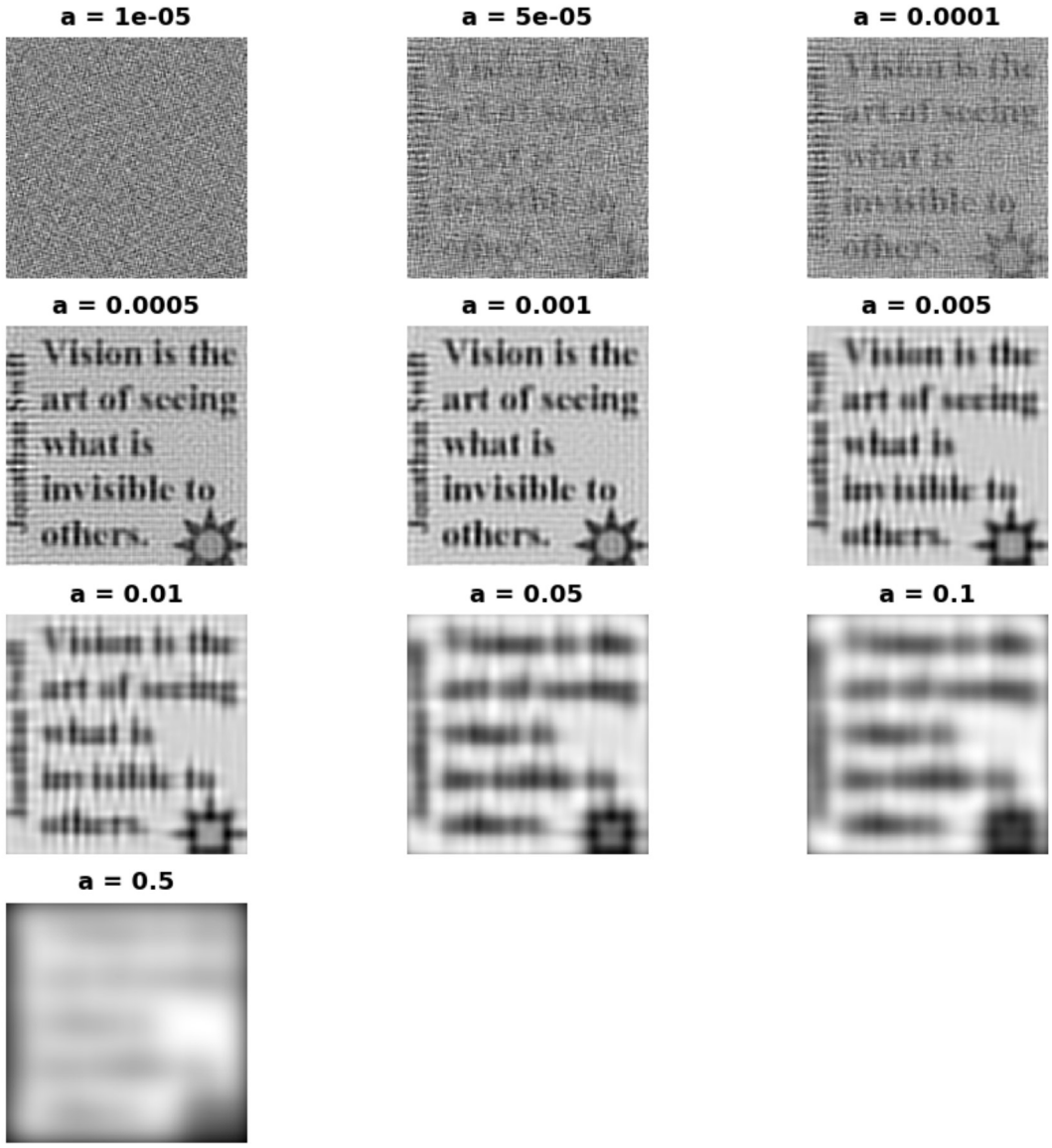
$$\hat{f}_i=\frac{\sigma_i\hat{g}_i}{\sigma_i^2+\alpha^2}$$

Parallel computation of Tikhonov regularization

As with the case of Truncated SVD this code is guaranteed to run on *Linux* and *Windows 10*.

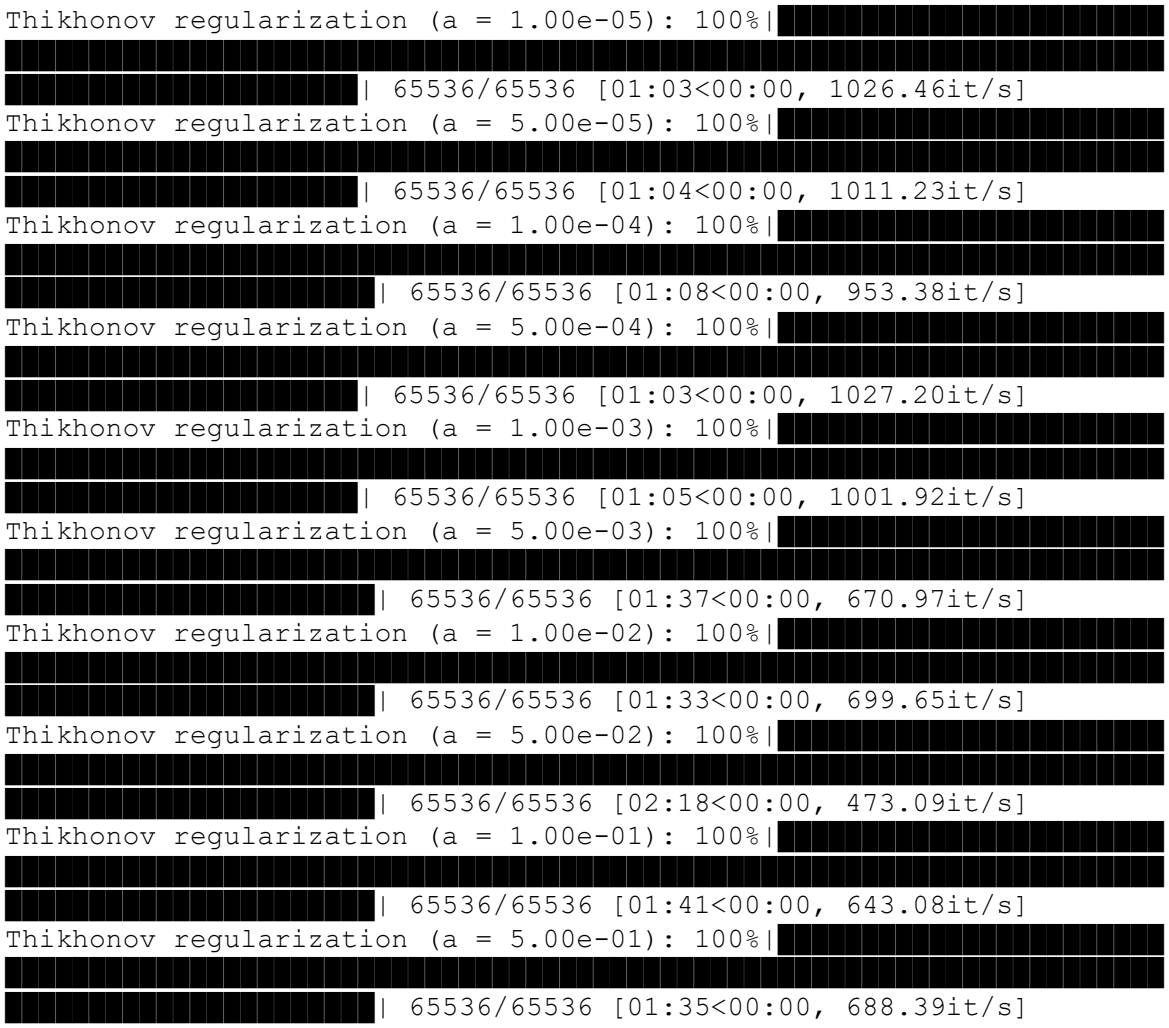
In [28]:
```
# Regularization parameters a.
a = [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1]

# Spawn tasks for parallel computation.
mp_runner_tikhonov_results = data_helper.mp_runner('Tikhonov', 'Tikhono
v', a, n, A, B, G)
```

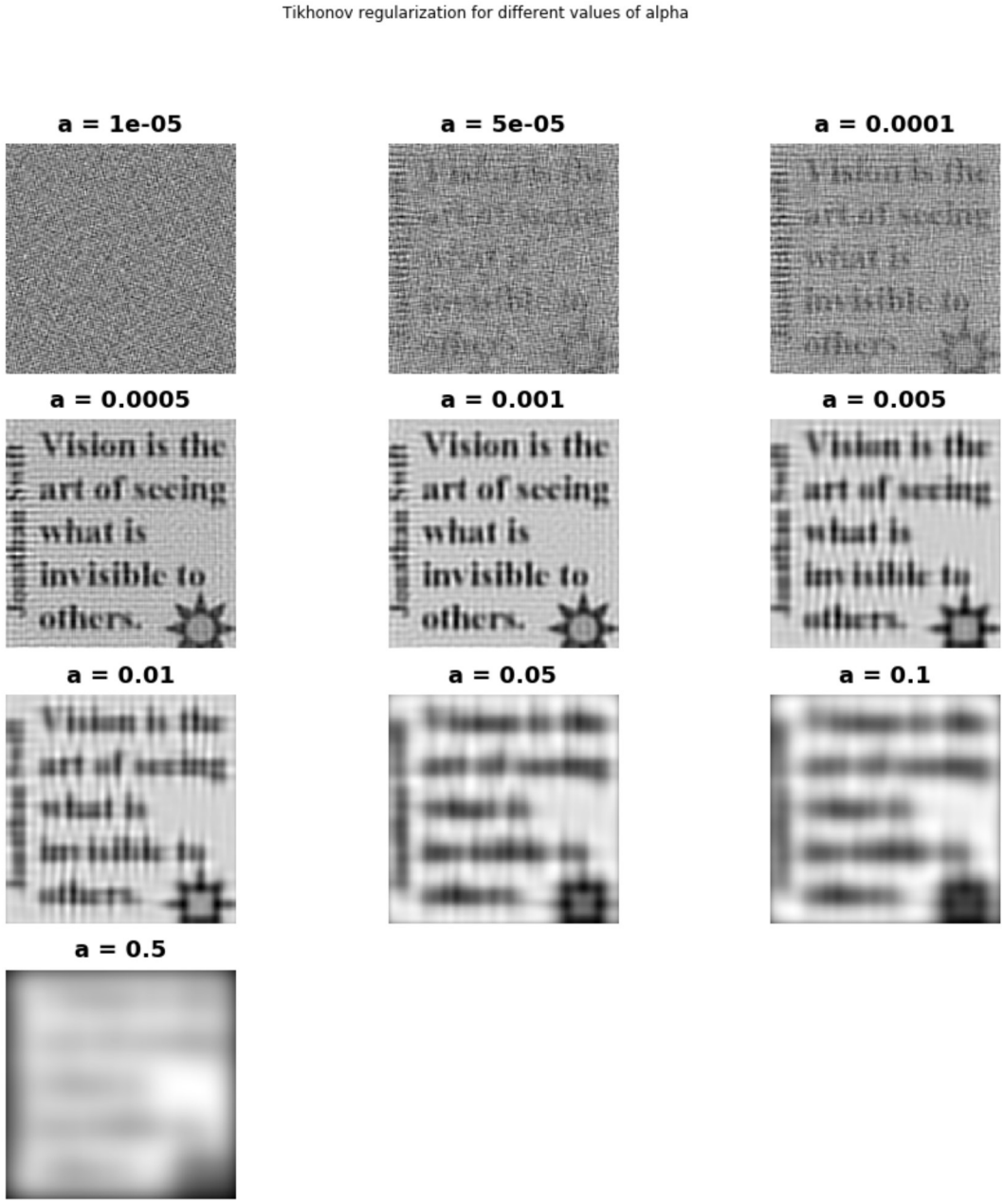
In [30]:
```
plot_main_title_font = {
    'family': 'sans serif',
    'color':  'black',
    'weight': 'bold',
    'size': 18
}

# Number of plot rows, columns
nrows = 4
ncols = 3

# Axis counters
ax_1 = 0
ax_2 = 0

# Setup the plot
fig, ax = plt.subplots(nrows = nrows, ncols = ncols, figsize = (15, 15)
)

fig.suptitle('Tikhonov regularization for different values of a', fontd
ict = plot_main_title_font)

for i in range(0, len(mp_runner_tikhonov_results)):
    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

    # Set graph title
    ax[ax_1, ax_2].set_title(label = 'a = ' + str(a[i]), loc = 'center'
, fontdict = plot_main_title_font, pad = 10)

    ax[ax_1, ax_2].imshow(mp_runner_tikhonov_results[i].F, cmap = plt.c
m.gray)

# Hide unused axes.
for i in range(len(a), nrows * ncols):
    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

pass;
```



Serial computation of Tikhonov regularization

In [77]:
```
from Tikhonov import Tikhonov

plot_main_title_font = {
    'family': 'sans serif',
    'color':  'black',
    'weight': 'bold',
    'size': 18
}

a = [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1]

# Number of plot rows, columns
nrows = 4
ncols = 3

# Axis counters
ax_1 = 0
ax_2 = 0

# Setup the plot
fig, ax = plt.subplots(nrows = nrows, ncols = ncols, figsize = (15, 15)
)

fig.suptitle('Tikhonov regularization for different values of alpha', f
ontdict = plot_main_title_font)

# f_hat_norm = []
# g_hat_norm = []

tikhonov = Tikhonov(with_progress_bar = True)

for i in range(0, len(a)):
    tikhonov = tikhonov.solve(n = n, A = A, B = B, G = G, a = a[i])

#     f_hat_norm.append(np.linalg.norm(tikhonov.f_hat, 2))
#     g_hat_norm.append(np.linalg.norm(tikhonov.g_hat, 2))

    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')

    # Set graph title
    ax[ax_1, ax_2].set_title(label = 'a = ' + str(a[i]), loc = 'center'
, fontdict = plot_main_title_font, pad = 10)

    ax[ax_1, ax_2].imshow(tikhonov.F, cmap = plt.cm.gray)

# Hide unused axes.
for i in range(len(a), nrows * ncols):
    if(i > 1 and i % ncols == 0):
        ax_1 += 1
    ax_2 = (i % ncols)

    ax[ax_1, ax_2].axis('off')
```


Thikhonov regularization (a = 1.00e-05): 100%| 65536/65536 [01:03<00:00, 1026.46it/s]
Thikhonov regularization (a = 5.00e-05): 100%| 65536/65536 [01:04<00:00, 1011.23it/s]
Thikhonov regularization (a = 1.00e-04): 100%| 65536/65536 [01:08<00:00, 953.38it/s]
Thikhonov regularization (a = 5.00e-04): 100%| 65536/65536 [01:03<00:00, 1027.20it/s]
Thikhonov regularization (a = 1.00e-03): 100%| 65536/65536 [01:05<00:00, 1001.92it/s]
Thikhonov regularization (a = 5.00e-03): 100%| 65536/65536 [01:37<00:00, 670.97it/s]
Thikhonov regularization (a = 1.00e-02): 100%| 65536/65536 [01:33<00:00, 699.65it/s]
Thikhonov regularization (a = 5.00e-02): 100%| 65536/65536 [02:18<00:00, 473.09it/s]
Thikhonov regularization (a = 1.00e-01): 100%| 65536/65536 [01:41<00:00, 643.08it/s]
Thikhonov regularization (a = 5.00e-01): 100%| 65536/65536 [01:35<00:00, 688.39it/s]



Several method have been described in the literature for choosing a good parameter for α, including the L-Curve (source: http://www2.compute.dtu.dk/~pcha/DIP/OLD/chap5.pdf (http://www2.compute.dtu.dk/~pcha/DIP/OLD/chap5.pdf)).

However, for the scope of this assignment, our "eyeball" estimation of the best parameter α appears to be 5e-4.
